2. Transactions

2. Transactions

2. Transactions

Transactions in Multi-User Mode

Run-time integrity

- Avoid errors resulting from simultaneous access of multiple users to the same data
- Multiple programs running simultaneously
  $\rightarrow$ concurrent and competing processes
- *Transaction* as a processing unit

Example scenarios

- Almost simultaneous seat reservations for flights from many travel agencies
  $\rightarrow$ Place could be sold several times when several travel agencies identify the space as available
- Overlapping operations of a bank account
- Statistical database operations
  $\rightarrow$ Results are falsified, if data is changed during the calculation

Transaction

A transaction is a sequence of operations (actions), which transfers a database from a consistent state into another eventually changed consistent state, applying the ACID properties.

ACID Properties

- **A**tomicity:
  Transactions are either completed, or not performed at all.
- **C**onsistency:
  If the database is in a consistent state before a transaction starts, the database is also consistent after the transaction has ended.
- **I**solation:
  A user who is working on the database should not notice any other user working on it.
- **D**urability:
  The result of a transaction must be permanently stored within the database, after the transaction is completed.

Commands of a transaction language

- Beginning of a transaction: Begin-of-Transaction-Command **`BOT`** (in SQL implicitly)
- **`commit`**: The transaction should be completed successfully
- **`abort`**: The transaction must be aborted

Problems in multi-user mode

- Non-repeatable read
- Read of inconsistent states
- Dirty read
- Phantom Problem
- Lost update
- Multi-user anomaly
- Problems with cursor references

Non-repeatable Read: Example I

- Constraint $X = A + B + C$ at the end of the transaction T_1
- X and Y are local variables
- T_i is the transaction i
- Integrity constraint: $A + B + C = 0$

Non-repeatable Read: Example II

T_1	T_2
$X := A;$	
	$Y := A/2;$
	$A := Y;$
	$C := C + Y;$
	commit;
$X := X + B;$	
$X := X + C;$	
commit;	

Read of Inconsistent States: Example

Integrity constraint $X + Y = 0$

T_1	T_2
read(X);	
read(Y);	
$X := X - 1$;	
write(X);	
$Y := Y + 1$;	
	read(X);
	read(Y);
write(Y);	
commit;	

Dirty Read: Example

T_1	T_2
read(X);	
$X := X + 100$;	
write(X);	
	read(X);
	$Y := Y + X$;
	write(Y);
	commit;
abort;	

Lost Update: Example

T_1	T_2	X
read(X);		10
	read(X);	10
$X := X + 1$;		10
	$X := X + 1$;	10
write(X);		11
	write(X);	11

The Phantom Problem: Example

T_1	T_2
select count (*) **into** X **from** employee **where** dept='CS'; **update** employee **set** Bonus = Bonus+10000/X **where** dept='CS'; **commit**;	**insert** **into** employee **values** (6789, 'Lilo', 'Pause', 'CS'); **commit**;

Multi-user anomaly: Example I

- Integrity constraint $A = B$

$$\begin{aligned} T_1 &:= \langle A := A + 10; B := B + 10 \rangle \\ T_2 &:= \langle A := A \cdot 1.1; B := B \cdot 1.1 \rangle \end{aligned}$$

- Each T_1 and T_2 keeps the IC isolated

Multi-user anomaly: Example II

T_1	T_2	A	B
read(A);		10	10
$A := A + 10$;			
write(A);		20	
	read(A);		
	$A := A \cdot 1.1$;		
	write(A);	22	
	read(B);		
	$B := B \cdot 1.1$;		
	write(B);		11
read(B);			
$B := B + 10$;			
write(B);		22	21

Problems with cursor references: Example

T_1	T_2
Position cursor C_1 on next tuple with property P (Tuple A)	
	Change property $P \rightarrow P'$ of A
Read current tuple	

Problems with cursor: SQL-notation

T_1	T_2
declare C_1 **cursor for** **select** * **from** Product **where** Price < 100; ... **fetch** C_1 **into** ...	 **update** Product **set** Price = 100 **where** ProdNr = 42;

Operations: user commands

- User commands that may influence the aborting of a transaction:
  - **`commit`** attempts to perform a commit $\rightsquigarrow$ not always successful (integrity violation) $\rightsquigarrow$ commit is only guaranteed if DBMS confirms the commit
  - **`abort`** forces final abort of a transaction
  - Other database operations that may lead to an abort: division by zero, integrity violations, ...

Operations: Internal

- **`commit`** performs a final commit internally
- Two versions for **`abort`**:
  - **`abort+restart`** aborts the transaction but also attempts to successfully complete the transaction by performing a restart (by intervention of scheduler; if **`abort+restart`** fails repeatedly, final abort is possible as well)
  - **`abort+stop`** performs the final abort (due to explicit user request or due to irreparable integrity violations)

Operations: Relation